L’ALGORITHMIQUE

Définitions générales

Informatique (computer science)

Sciences et techniques de traitement automatique de l'information

Ordinateur

Calculateur électronique capable d'effectuer une succession de tâches définies par l'utilisateur

Algorithme

Définition univoque de la succession de tâches permettant d'obtenir un résultat en un nombre fini d'étapes

Définitions générales

Langage de programmation

Code permettant de décrire des algorithmes et leurs données sous une forme permettant leur exécution par une machine, ET leur compréhension par des lecteurs humains.

Programme

codage d'un algorithme dans un langage de programmation

Définition d'un Algorithme

- Algorithmique: Science qui étudie l'application des algorithmes à l'informatique
- Algorithme: Procédure décrivant, étape par étape, une méthode permettant de résoudre un problème.
- C'est la base de tout programme informatique

Exemple: Changer une roue:

1. Sortir cric, manivelle, et roue de secours
2. Monter la voiture sur le cric
3. Démonter la roue
4. Monter la roue de secours
5. Enlever le cric
6. Ranger cric, manivelle, et roue de secours

Algorithmique:Etapes

- Conception de méthodes pour la résolution de problèmes
  - On a une description des données d'un problème (entrées, spécification en mots du résultat cherché)
  - On décrit des métodes pour résoudre le problème: ALGORITHME
  - On montre que les méthodes répondent au problème
- Complexité des méthodes
  - Efficacité: temps de calcul, espace mémoire utilisé
- Réalisation
  - Organisation des données: Choix d'une structure de données
  - Implémentation

Caractéristiques d'un algorithme

- L'algorithme est un moyen pour le programmeur de présenter son approche du problème à d'autres personnes.
- Un algorithme est l'énoncé dans un langage bien défini d'une suite d'opérations permettant de répondre au problème.

Caractéristiques d'un algorithme

- Un algorithme doit donc être :
  - **Lisible:** compréhensible même par un non-informaticien
  - **De haut niveau:** traduit en n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné
  - **Précis:** chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important de lever toute ambiguïté
  - **Concis:** ne doit pas dépasser une page. Si c'est le cas, il faut décomposer le problème en plusieurs sous-problèmes
  - **Structuré:** composé de différentes parties facilement identifiables.

Caractéristiques d'un algorithme

Suite finie d'étapes (instructions) vérifiant:

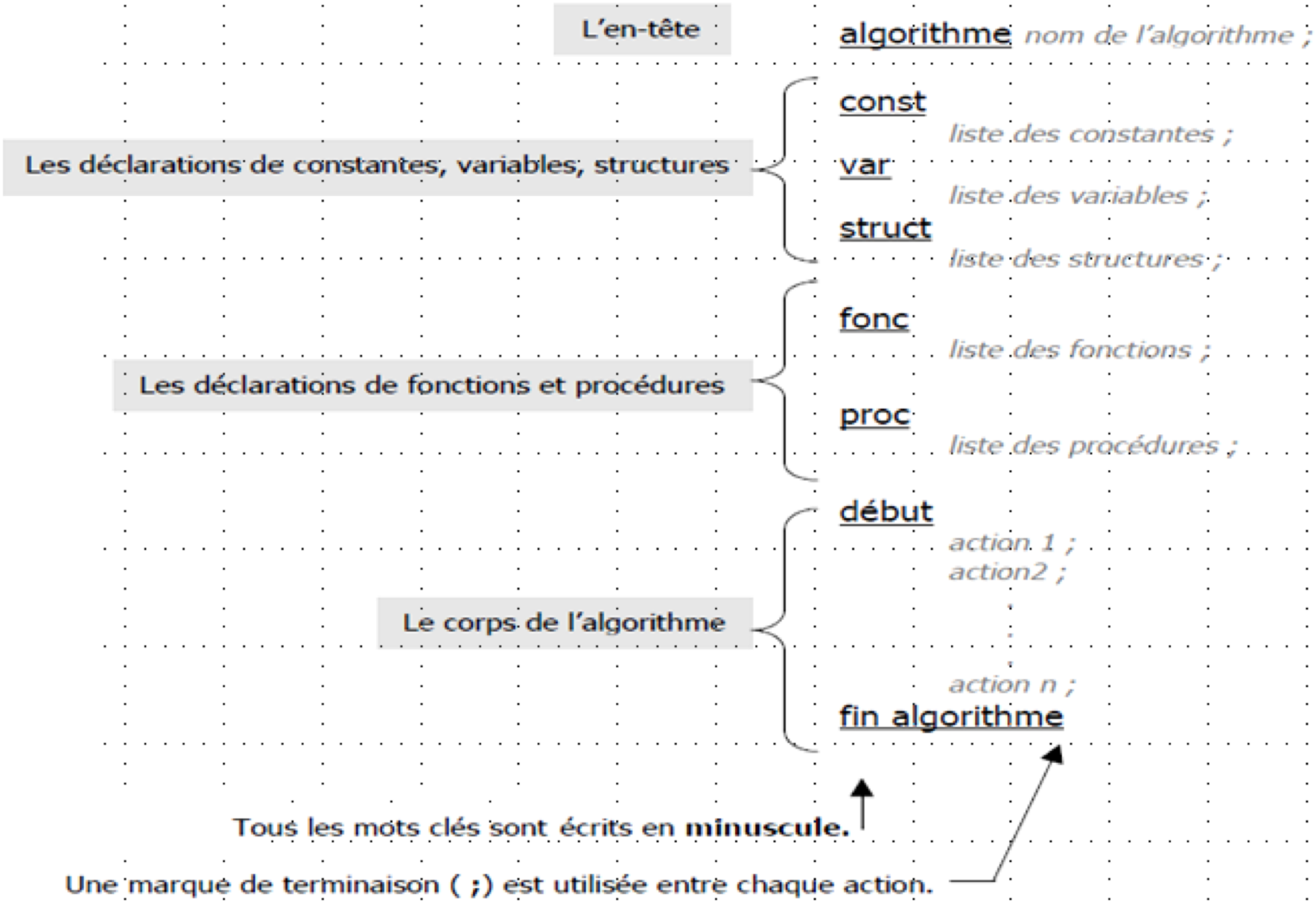
- Chaque étape est décrite de façon précise;
- Chaque étape est déterministe: produit des résultats uniques;
- L'algorithme s'arrête après un nb fini d'instructions
- Reçoit des données en entrée;
- Produit des données en sortie;

- Généralité: Applicable à des ensembles différents de données d'entrée

Conventions d'écriture

- une représentation graphique, avec des carrés, des losanges, etc. qu'on appelait des **organigrammes**
- « **pseudo-code** » ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe.

Structure d'un algorithme



Structure d'un algorithme

1. L'en-tête: il permet tout simplement d'identifier un programme.
2. Les déclarations: c'est une liste exhaustive des objets, grandeurs utilisés et manipulés dans le corps de l'algorithme. Cette liste est placée en début du programme.
3. Le corps: dans cette partie de l'algorithme sont placées les tâches (instructions, opérations …) à exécuter .
4. Les commentaires: pour permettre une interprétation aisée de l'algorithme, l'utilisation des commentaires est vivement conseillée.

Déclaration de constantes, de variables et de structures.

a. **Les constantes:** Elle représentent des chiffres, des caractères, des chaînes de caractères,…. C'est une donnée qui ne peut pas être modifiée au cours de l'exécution de l'algorithme.

Syntaxe: constante Nom constante:[type]= valeur

Exemple: Constante Pi:réel=3.141559

Constante Nombre Lettre: entier=10

b. **Les variables**: Elles peuvent stocker des chiffres des nombres, des caractères, des chaînes de caractères,… C'est une donnée qui peut être modifiée au cours de l'exécution de l'algorithme.

Déclaration de constantes, de variables et de structures.

- Syntaxe: variable nom variable:[Type]
  - Exemples: variable rayon: Réel

    variable compteur: Entier

    variable lettre: caractère

c. Les structures: Elles permettent de rassembler plusieurs variables ou constantes sous un même identificateur, on parle aussi d'entités ou d'objets.

Déclaration de procédures et de fonctions.

- **La procédure**: c'est un ensemble d'instructions référencé par un nom, et dont l' « exécution » est provoquée par le simple énoncé de ce nom. Les procédures sont des sous-programmes qui ne retournent **aucun résultat**
- Syntaxe de la déclaration d'une procédure:

procédure Nom procédure (Nomentrée1:[type], Nom entrée 2:[Type],…)

constante ~ déclaration des constantes locales~

variable ~ déclaration des variables locales~

début

~ description des actions effectuée par la procédure ~

fin

Déclaration de procédures et de fonctions.

- **La fonction**: comme pour la procédure, l'exécution d'une fonction est provoqué par la simple évocation de son nom, à la différence qu'elle se voit assigner une valeur dont le type doit être défini. Les fonctions sont des sous-programmes admettant des paramètres et retournant un **seul résultat**
- Syntaxe de la déclaration d'une fonction:

fonction NomFonction (Nomentrée1:[type], Nomentrée2:[Type],…): type de la valeur retournée

constante ~ déclaration des constantes locales~

variable ~ déclaration des variables locales~

début

~ description des actions effectuée par la fonction~

fin

Exemples d'appels de fonctions et procédures

- Procédure sans paramètre :
  - Ex : Effacer_Ecran
- Procédure avec un paramètre d'entrée :
  - Ex : Afficher ('Bonjour')
- Fonction avec paramètres d'entrée et de sortie :
  - Ex : Resultat ⟵ Racine (69)

Exemple de déclaration de fonction

- **Fonction Moyenne (Note1 : Reel, Note2 : Reel) : Reel**
  Variable Intermediaire : Reel
  Début
  Intermediaire ⟵ Note1 + Note2;
  Intermediaire ⟵ Intermediaire / 2;
  Moyenne ⟵ Intermediaire;
  Fin
- Exemples d'utilisation de la fonction :
  - Afficher (Moyenne(10.5,15));
  - NouvelleNote⟵ Moyenne (10,5.5);

Programme principal

- Le programme principal consiste en une suite d'opérations élémentaires faisant souvent appel à des fonctions ou procédures. Ces différentes opérations sont mentionnées en utilisant les structures algorithmiques décrites précédemment.
- Le programme principal est délimité par les mots clefs **début** et **Fin**.

Les types de base

- **Entiers**: valeurs numériques entières pouvant être signées ou non signées (codées sur un ou plusieurs octets).
  - Mot clé: entier
- **Réels**: valeurs numériques codées avec une mantisse et un exposant.
  - Mot clé: réel
- **Booléennes**: valeur pouvant être soit Vraie, soit Fausse.
  - Mot clé: booléen
- **Caractère**: octet correspondant à un code ASCII.
  - Mot clé: car
- **Chaîne de caractères**: ensemble de caractères.
  - Mot clé: chaîne
- **Tableau de données**: ensemble de données de même type (tableau d'entiers, tableau de réels).

Les opérateurs

- Opérateurs sur les entiers et les réels:

Arithmétiques	
+	Addition
-	Soustraction
*	Multiplication
/	Division
DIV	Division entière
↑	Puissance

comparaisons	
>	Supérieur
<	Inférieur
≥	Supérieur ou égal
≤	Inférieur ou égal
=	Egal
≠	Différent

Les opérateurs

- Opérateurs sur les entiers et les booléens:

Fonctions logiques	
Mot clé	
et	Fonction ET
ou	Fonction OU
oux	Fonction OU exclusif
non	Fonction NON
non et	Fonction NON ET
non ou	Fonction NON OU

Fonctions de comparaison pour les booléens	
=	Egal
≠	Différent

Les opérateurs

- les opérateurs sur les caractères et les chaînes:

Fonctions de concaténation	
+	Concaténation

Fonctions de comparaison pour les chaînes	
=	Egalité
≠	Différent
>	Supérieur
<	Inférieur

- Remarque:
  - Priorité des opérateurs: Priorité à la multiplication et à la division.

Les opérateurs

- **Conditions**: dans les structures qui vont suivre, le terme «condition » peut représenter:
  - Une condition simple :Ex: x≠0 Indice ≥80
  - Une condition complexe:

  (Indice ≥1) ET (Indice ≤10) ~ pour 1≤Indice ≤10 ~
- **L'affectation**: Elle permet d'affecter une valeur à une variable.
  - Syntaxe: identificateur de la valeur ⟵ expression

    Ou bien :=

Les opérateurs

- L'expression est une suite d'opérations sur les constantes ou les variables déjà déclarées « expression » peut être:
  - Une constante: Ex: surface ⟵ 40
  - Une autre variable: Ex: Donnée ⟵ valeur mémorisée.
- L'affectation se fait toujours en deux temps:
  1. Evaluation de l'expression située à droite du symbole
  2. Affectation du résultat à l'identificateur de variable

y ⟵ z signifie ``copie la valeur de z dans y''.
Valeur de z inchangée

Structures Algorithmique

- Exécution séquentielle
  - Une instruction par ligne
  - Exécution du haut vers le bas

Structures Algorithmique

- Exécution conditionnelle
  - exécuter (ou non) des instructions en fonction du résultat d'un test
  - syntaxe simple :

```
si <expression booleenne> alors
    <instructions>
fin si
```

  - syntaxe avec sinon :

```
si <expression booleenne> alors
    <instructions>
sinon
    <instructions>
fin si
```

Structures Algorithmique

- Exécution itérative
  - répéter une série d'instruction tant qu'un test est vrai
  - syntaxe

```
tant que <expression booleenne> faire
    <instructions>
fin tant que
```

Les opérateurs

Exemple

Variable Bonjour en Caractère Début Bonjour ← "Ave Cesar" Ecrire Bonjour Fin	Variable Bonjour en Caractère Début Bonjour ← "Ave Cesar" Ecrire "Bonjour" Fin

Le premier algorithme affiche le contenu de la variable Bonjour, autrement dit " Ave Cesar ". Le second affiche le libellé " Bonjour ", qui n'a rien à voir avec la variable du même nom.

Différence entre problème et instance du problème

- Exemple d'un problème: Tri d'un ensemble d'éléments
  - Entrée: Suite de *n* éléments $a_1,\ldots a_n$
  - Sortie: La suite ordonnée
- Instances du problème: entrées nécessaires pour calculer une solution du problème
  - Suite de nombres: *475, 787, 34, 245, 56, 350*
  - Suite de noms: *Pierre, Christine, Sylvie, Samuel, Fabien*

Structures de choix (ou conditionnelles)

i. **Structure si……Alors…**

Une condition est testée pour déterminer si l'action ou le groupe d'actions suivant doit être exécuté.

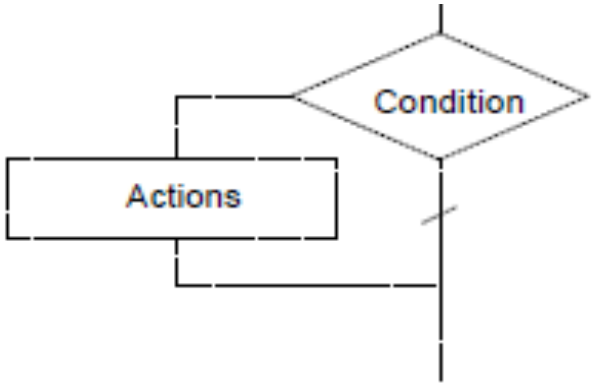
Syntaxe:

si condition

Alors Actions

Finsi

Organigramme



Exemple

Exemple: calcul d'une racine carrée
Algorithme calcul d'une racine carrée
variables
x: réel;
r: réel;
Début
écrire ('donnez le nombre x');
lire (x);
Si x>0 **Alors**
r⟵racine (x);
écrire (r);
Finsi
Fin

Pseudo-Code

Algorithme maximum:

// Retourne le maximum de 3 entiers

- Entrée: Trois entiers a, b, c;
- Sortie: x contenant la valeur maximale
- ***Procédure max(a:entier,b:entier,c:entier)***
- **x:entier**

1. ***début***
2. *x := a;*
3. **Si** *b>x* **alors** //Si b plus grand que x, mettre x à jour
4. *x := b;* **FinSi**
5. **Si** *c>x* **alors** // Si c plus grand que x, mettre x à jour
6. *x := c;* **FinSi**
7. **écrire** (x);
8. **Fin**

Trace de l'algorithme pour:

a=1; b=5; c=3

x = 1

x = 5

Instruction Si-Alors-Sinon

- Une condition est testée pour déterminer quelle action ou quel groupe d'actions doit être exécuté.
- **Syntaxe:**

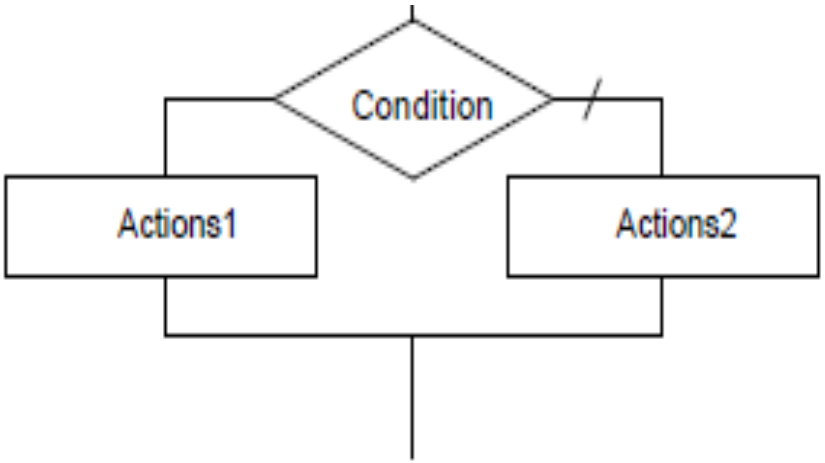
si condition

Alors Action1

Sinon action2

Finsi

Organigramme



Exemple

Exemple: calcul d'une racine carrée
Variable
X: réel;
r: réel;
Début
écrire ('donner le nombre x');
lire (x);
Si x<0 **Alors**
afficher ('x est négatif');
sinon
r ⟵ racine (x);
écrire (r);
Finsi
Fin

Exemple

Exemple: Calcul le maximum de deux entiers
Variable
entier1 : entier;
entier2: entier;
Max: entier;
Début
écrire (‘donnez l’entier1’);
lire (entier1);
écrire (‘donnez l’entier2’);
  lire (entier2);

si entier1<entier2 alors
Max ← entier2;
sinon
Max← entier1;
Finsi
Ecrire(max);
fin

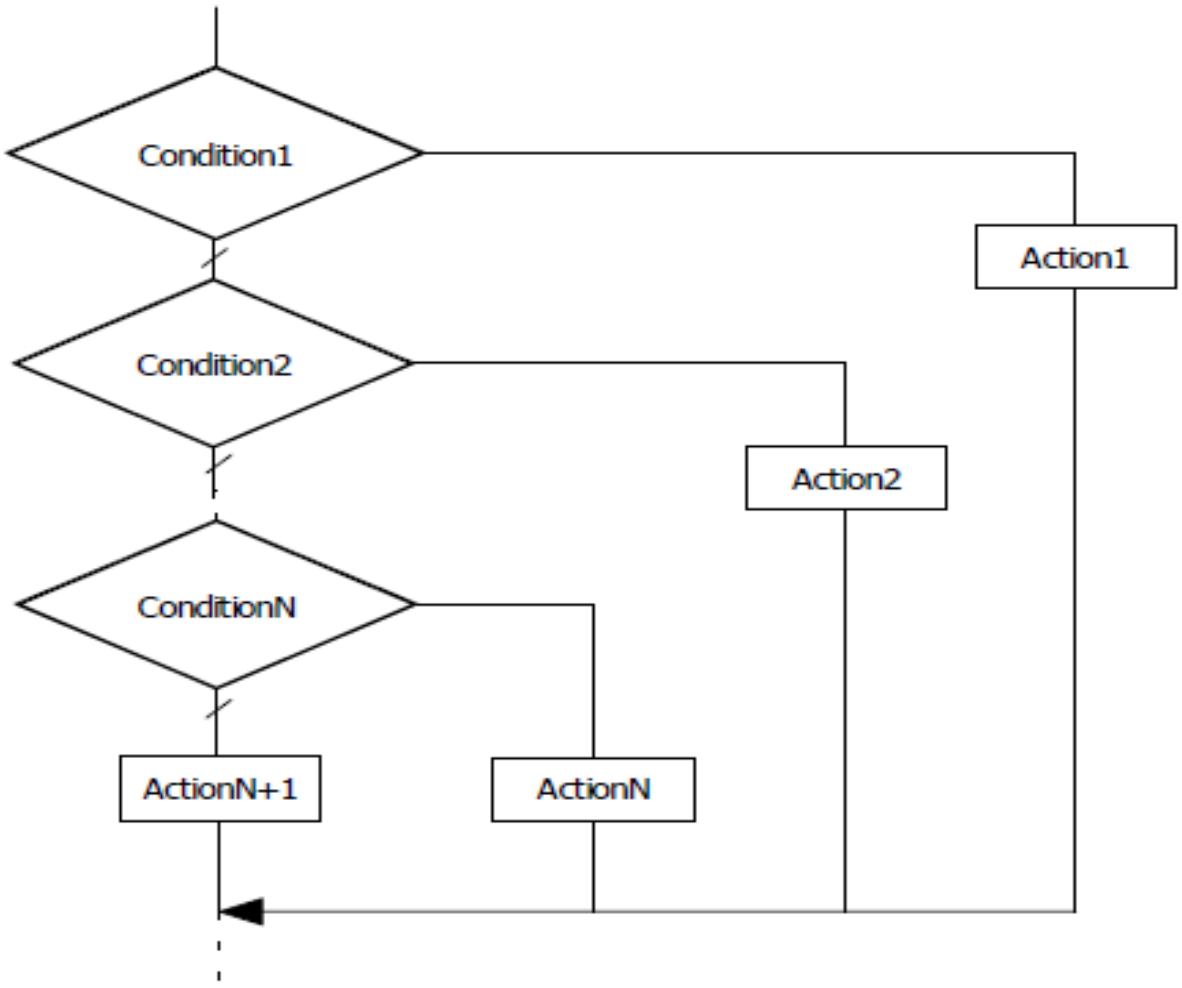
Structure de choix multiple

- La structure de choix permet, en fonction de plusieurs conditions de type booléen, d'effectuer des actions différentes suivant les valeurs que peut prendre une même variable.
- Syntaxe:

```
suivant valeur faire
valeur1 : action1 ;
valeur2 : action2 ;
 :
 :
valeurN : actionN ;
sinon actionN+1 ;
fsuivant ;
```

Structure de choix multiple

Organigramme

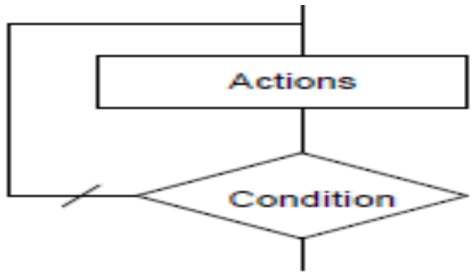


Structures itératives (ou répétitives)

- **Structure REPETER……JUSQUA…**

  Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une condition est vérifiée.

Organigramme



Syntaxe:

Répéter

Actions

Jusqu'à condition

Remarque: la vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins une fois.

Exemple: Exécution répétitive d'un programme:

Variables:
a,b:réel;
p:réel;
c:caractère;
Début
Répéter
écrire ('donnez le nombre a');
lire (a);
ecrire('donnez le nombre b');
lire (b);
P ⟵ a*b;
ecrire(p);
ecrire('encore un calcul? Non touche N; oui autre touche');
lire (c);
Jusqu'à c='N'
Fin

Structures itératives (ou répétitives)

- **Structure TANT QUE……FAIRE…**

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est vraie.

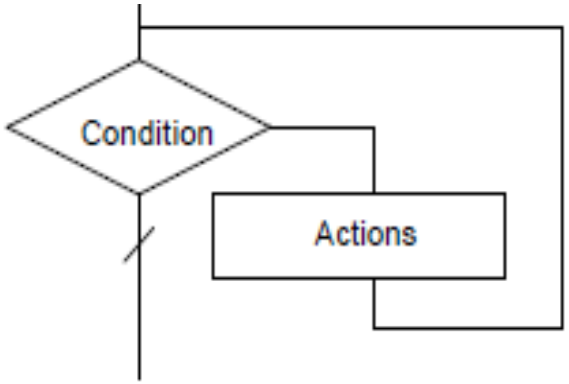
Syntaxe:

Tant que condition

Faire Actions

Fin tantque

Organigramme



Exemple: exécution répétitive d'une action

Remarque: la vérification de la condition s'effectue avant les actions. celles-ci peuvent donc ne jamais être exécutées.

Variables:
x:entier;
y:réel;

- **Début**

```
écrire ('donnez le nombre x');
lire (x);
tant que x>0
        Faire
                y:=racinecarré(x);
            écrire(y);
        Fin tant que
Fin
```

Instruction Tant que

Algorithme Plus-Grand-Element: Retourne la plus grande valeur d'une liste

Entrée: *n* entiers $S_1,\ldots, S_n$

Sortie: le plus grand élément

début

grand := S_1;

i:=2;

Tant que *i* ≤ *n* **Faire**

Début

Si S_i > *grand* **alors** // une plus grande valeur a été trouvée

grand := S_i;

i := i+1;

FinFaire

afficher *(grand)*

Fin

Trace de l'algorithme:

n=4; S_1= -2; S_2=6; S_3=5; S_4=6

grand =-2	*i* = 2
grand = 6	*i* = 3
	i = 4
	i = 5

Structures itératives
Instruction Pour (For)

Pour *var := init* **à** *limite* **Faire**

action;

À chaque passage dans la boucle, *var* est incrémenté de 1. On s'arrête dès que *var > limite*

Algorithme Plus-Grand-Element: Réécriture de l'algorithme précédent mais avec une boucle ``Pour''

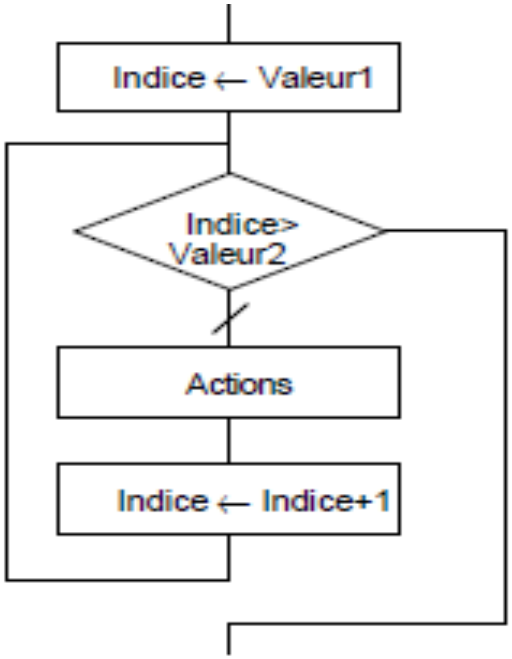
Entrée: *n* entiers S1,…, Sn

Sortie: *grand* contenant le plus grand élément

Procédure *plus-grand (S,n)*
grand := S_1;
Pour *i =1* **à** *n* **Faire**
  Si *S_i > grand* **alors** // une plus grande valeur a été trouvée
    grand := S_i;
ecrire(*grand*)
Fin *plus-grand;*

Structures itératives
Instruction Pour (For)

Organigramme



Exemple: Affichage d'une ligne d'étoiles

Variables:
i:entier;
Début
pour i := 1à 80
Faire
Afficher('*');
Finpour
Fin

Structures itératives

- *Remarque*: cette structure algorithmique peut être remplacée par une structure TANT QUE….FAIRE…
- Variable

```
i:entier
Début
    i⟵ 1
Tant Que i≤80
    Faire
        Afficher (‘*’)
        i⟵ i+1
    Fin tantque
Fin
```

Exemple 1

Équation du second degré dans $\mathbb{R}$

```
variable a,b,c en flottant
variable delta,x1,x2 en flottant
Lire a
Lire b
Lire c
delta ← b*b-4*a*c
si delta < 0 alors
    Ecrire "pas de solution
sinon
    si delta > 0 alors
        x1 ← (-b+deltaˆ0.5)/(2*a)
        x2 ← (-b-deltaˆ0.5)/(2*a)
        Ecrire "deux solutions : " + x1 + " et " + x2
    sinon
        Ecrire "une solution double : " + (-b/(2*a))
    fin si
fin si
```

Exemple 2

- PGCD: pgcd(12,8)
- pgcd(u,v) : plus grand entier divisant u et v
- le pgcd est plus petit que u et que v

8 : ne divise pas 12
7 : ne divise ni 12 ni 8
6 : ne divise pas 8
5 : ne divise ni 12 ni 8
4 : divise 12 et 8

Exemple 2: PGCD

entiers relatifs

$pgcd(u, v) = pgcd(|u|, |v|)$

- il suffirait donc de calculer les valeurs absolues

le cas d'un zéro

- si u est nul et v non nul, $pgcd(u, v) = v$
- si u est non nul et v est nul, $pgcd(u, v) = u$
- ces cas particuliers devraient être testés en début d'algorithme

le cas de deux zéros

- si u et v sont nuls tous les deux leur pgcd n'est pas défini
- dans ce module, nous supposons que les valeurs entrées sont valides (pas de gestion des erreurs)

Exemple 2: PGCD

```
variable u,v en entier
variable p en entier
Lire u
Lire v
si u < 0 alors
    u ← -u
fin si
si v < 0 alors
    v ← -v
fin si
si u == 0 alors
    p ← v
sinon
    si v == 0 alors
        p ← u
```

Exemple 2: PGCD

```
    sinon
        si u < v alors
            p ← u
        sinon
            p ← v
        fin si
        tant que (u mod p ≠ 0) ou (v mod p ≠ 0) faire
            p ← p-1
        fin tant que
    fin si
fin si
Ecrire "pgcd : " + p
```

Tableaux

Les tableaux sont des objets élémentaires du langage, avec des opérations spécifiques. Ils auraient donc du être introduits dans la partie "opérations et objets élémentaires". Nous avons décalé leur présentation pour permettre une meilleure compréhension

But

- représenter un "ensemble" de valeurs de même type
- chaque valeur est identifiée dans le tableau par un indice

Tableaux $\Longleftrightarrow$ Vecteurs

Les tableaux informatiques sont très proches de la notion de vecteur en mathématique.

informatique

tableau de 5 flottants :

2.0	-1.2	2.0	0.0	5.3
↑	↑	↑	↑	↑
0	1	2	3	4

indices :

mathématique

- $\boldsymbol{X} \in \mathbb{R}^n$
- $\boldsymbol{X} = (x_1, x_2, ..., x_n)$; $x_i \in \mathbb{R}$

Le plus souvent, les indices commencent à 0 en informatique, et à 1 en mathématique.

Utilisation des Tableaux

1. déclaration : type tableau

   tableau de <type des éléments>

2. création d'un tableau

   nouveau tableau de <nombre d'éléments> <type des éléments>

3. accès aux éléments

   <tableau> **[** <indice> **]**

4. retrouver la taille d'un tableau

   taille(<tableau> **)**

Tableaux: Exemple

```
variable t en tableau de flottant
variable i,n en entier
Ecrire "nombre d'éléments : "
Lire n
t ← nouveau tableau de n flottant
i ← 0
tant que i < n faire
    Ecrire "valeur de l'élément N°" + i + " : "
    Lire t[i]
fin tant que
i ← 0
tant que i < n faire
    si t[i] < 0 alors
        t[i] ← -t[i]
    fin si
fin tant que
```

Exercices

- ***Exercice 1***

Quelles seront les valeurs des variables A et B à la fin de l'algorithme suivant ?

Variables A, B : Entier *Début* *A ← 1* *B ← A + 3* *A ← 3* *Fin*	*Variables A, B : Entier* *Début* *A ← 5* *B ← A + 4* *A ← A + 1* *B ← A – 4* *Fin*
Variables A, B, C : Entier *Début* *A ← 3* *B ← 10* *C ← A + B* *B ← A + B* *A ← C* *Fin*	*Variables A, B, C, D : Caractères* *Début* *A ← 'A'* *B ← "12"* *C ← "ABC"* *Fin*

Exercices

- ***Exercice 2***

Quels sont les contenus des variables après l'exécution de chaque instruction: A, B, C sont des variables entières et cond une variable de type booléen

	A	B	C	Cond
A←1				
B ← 2				
C ← 3				
A ← A + B * C				
A ← A * B + C				
B ← C + 1				
B ← C * A - B * B				
C ← C / C				
Cond ← (A<B)				
Cond ← (A+B ≠ 0)				
Cond ← (cond) ET (C=10)				

Exercices

- ***Exercice 3***
- Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?
- **Variables** A, B **en Entier**
  Début
  A ← 5;
  B ← 2;
  A ← B;
  B ← A;
  Fin
- Moralité : les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

Corrigé

- Après — La valeur des variables est :

Après		
A ← 5	A = 5	B = ?
B ← 2	A = 5	B = 2
A ← B	A = 2	B = 2
B ← A	**A = 2**	**B = 2**

- Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée. Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

Exercices

- ***Exercice 4:***
- Plus difficile, mais c'est un classique absolu, qu'il faut absolument maîtriser :
- écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quelque soit leur contenu préalable.

Corrigé

- **Début**

  …
  C ← A
  A ← B
  B ← C
  Fin
- On est obligé de passer par une variable dite temporaire (la variable C).

Exercices

- ***Exercice 5:***
- Une variante du précédent : on dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

Corrigé

- **Début**
  …
  D ← C
  C ← B
  B ← A
  A ← D
  Fin
- En fait, quel que soit le nombre de variables, une seule variable temporaire suffit…

Exercices

- ***Exercice 6***
- Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit **pas** calculer le produit des deux nombres.

Corrigé

- **Algorithme: signe du produit**
- **Variables** m, n **en Entier;**
  Début
  Ecrire ("Entrez deux nombres :");
  Lire (m, n);
  Si (m > 0 ET n > 0) OU (m < 0 ET n < 0)
  Alors
  Ecrire ("Leur produit est positif");
  Sinon
  Ecrire ("Leur produit est négatif");
  Finsi
  Fin